An Empirical Evaluation of Portfolios Approaches for Solving CSPs

Roberto Amadini, Maurizio Gabbrielli, Jacopo Mauro

An Empirical Evaluation of Portfolios Approaches for solving CSPs

Roberto Amadini, Maurizio Gabbrielli, and Jacopo Mauro

Department of Computer Science and Engineering/Lab. Focus INRIA,
University of Bologna, Italy.
{amadini, gabbri, jmauro}@cs.unibo.it

Abstract. Recent research in areas such as SAT solving and Integer Linear Programming has shown that the performances of a single arbitrarily efficient solver can be significantly outperformed by a portfolio of possibly slower on-average solvers. We report an empirical evaluation and comparison of portfolio approaches applied to Constraint Satisfaction Problems (CSPs). We compared models developed on top of off-the-shelf machine learning algorithms with respect to approaches used in the SAT field and adapted for CSPs, considering different portfolio sizes and using as evaluation metrics the number of solved problems and the time taken to solve them. Results indicate that the best SAT approaches have top performances also in the CSP field and are slightly more competitive than simple models built on top of classification algorithms.

1 Introduction

The past decade has witnessed a significant increase in the number of constraint solving systems deployed for solving *Constraint Satisfaction Problems* (CSP). It is well recognized within the field of constraint programming that different solvers are better at solving different problem instances, even within the same problem class [3]. It has also been shown in other areas, such as satisfiability testing [18] and integer linear programming [9], that the best on-average solver can be out performed by a portfolio of possibly slower on-average solvers. This selection process is usually performed by using *Machine Learning* (ML) techniques based on feature data extracted from the instances that need to be solved. Thus in general a *Portfolio Approach* [3] is a methodology that exploits the significant variety in performances observed between different algorithms and combines them in a portfolio to create a globally better solver. Portfolio approaches in particular have been extensively studied and used in the SAT solving field. On the other hand, to the best of our knowledge in the CSP field there exists only one solver that uses a portfolio approach, namely CPHydra [13]. This solver uses a rather small portfolio (just 3 solvers) and seems rather limited when compared to modern SAT portfolio approaches.

In this work we tried to investigate to what extent a portfolio approach can increase the performances of a CSP solver and which could be the best portfolio approaches, among the several existing, for CSPs. We considered 22 versions of 6 well known CSP solvers and using these 22 solvers we implemented two classes of CSP portfolio solvers, building portfolios of up to 16 solvers: in the first class we used relatively simple, off-the-shelf machine learning classification

algorithms in order to define solver selectors; in the second class we tried to adapt the best, advanced, and complex approaches of SAT solving to CSP. A third portfolio solver that we considered was CPHydra, mentioned above. We then performed an empirical evaluation and comparison of these three different portfolio approaches. We hope that our results, described in the remaining of this paper, may lead to new insights, to a confirmation of the quality of some approaches and also to some empirical data supporting the creation of better and faster CSP solvers.

It is worth noticing that adapting portfolios techniques from other fields is not trivial: for instance, since portfolio approaches usually exploit features extracted from the various instances of the problems, a good features selection may be responsible of the quality and the performances of an approach. Moreover, differently from the SAT world, in the CSP field there is no a standard language to express CSP instances, there are fewer solvers, and sometimes only few features and constraints are supported. To overcome these limitations we tried to collect a dataset of CSP instances as extensive as possible. We used this dataset to evaluate the performances of the three different CSP portfolio approaches.

2 Preliminaries

In this section we describe CPHydra and the SAT specific portfolio approaches that we have adapted to CSP.

CPHydra To our knowledge CPHydra [13] is the only CSP solver which uses a portfolio approach. This solver uses a k-nearest neighbor algorithm in order to compute a schedule of the portfolio constituent solvers which maximizes the chances of solving an instance within a time-out of 1800 seconds. CPHydra was able to win the 2008 International CSP Solver Competition.

SAT Solver Selector (3S) 3S [6] is a SAT solver that conjugates a fixed-time static solver schedule with the dynamic selection of one long-running component solver. It first executes for 10% of its time short runs of solvers. The schedule of solvers, obtained by solving an optimization problem similar to the one tackled by CPHydra, is computed offline (i.e. during the learning phase on training data). Then, at run time, if a given instance is not yet solved after the short runs a designated solver is executed for the remaining time. This solver is chosen among the ones that are able to solve the majority of the most k-similar instances in the training dataset. 3S was the best-performing dynamic portfolio at the International SAT Competition 2011.

SATzilla SATzilla [18] is a SAT solver that relies on runtime prediction models to select the solver that (hopefully) has the fastest running time on a given problem instance. In the International SAT Competition 2009, SATzilla won all three major tracks of the competition. More recently a new powerful version of SATzilla has been proposed [17]. Instead of using regression-based runtime predictions, the newer version uses a weighted random forest approach provided with an explicit cost-sensitive loss function punishing misclassifications in direct proportion to their impact on portfolio performance. This last version consistently outperforms the previous versions of SATzilla and the other competitors of the SAT Challenge 2012 in the Sequential Portfolio Track.

ISAC In [10] the Instance-Specific Algorithm Configuration tool ISAC [7] has been used as solver selector. Given a highly parametrized solver for a SAT

instance, the aim of ISAC is to optimally tune the solver parameters on the basis of the given instance features. It can be easily seen as a generalization of an algorithm selector since it could be used to cluster the instances and when a new instance is encountered it selects the solver that solved the largest number of instances belonging to the nearest cluster.

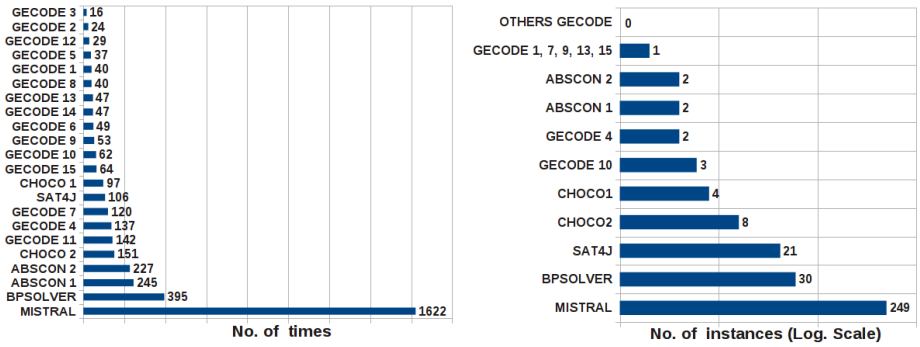
3 Solvers, Features and Dataset

In this section we introduce the three main ingredients of our portfolios: the CSP solvers that we use; the features, extracted from the CSP instances, which are used in the machine learning algorithms; the dataset used to perform the tests.

Solvers We decided to build our portfolios by using some of the solvers of the International CSP Solver Competition. We were able to use 5 solvers of this competition, namely AbsCon (2 versions), BPSolver, Choco (2 versions), Mistral and Sat4j. Moreover, by using a specific plug-in described in [11], we were able to use also 15 different versions of the constraint solver Gecode (these different versions were obtained by tuning the search parameters and the variable selection criteria of the solver). Thus we had the possibility of using, in our portfolio, up to 22 specific solvers which were all able to process CSP instances defined in the XCSP format [14].

Features In order to train the classifiers, we extrapolated a set of 44 features from each XCSP instance. An extensive description of the features can be retrieved in [8]. We used the 36 features of CPHydra [13] plus some features derived from the *variable graph* and *variable-constraint graph* of the XCSP instances. Whilst the majority of these features are syntactical, some of them are computed by collecting data from short runs of the Mistral solver.

Dataset We tried to perform our experiments on a set of instances as realistic and large as possible. Hence, we constructed a comprehensive dataset of CSPs based on the instances gathered from the 2008 International CSP Solver Competition that are publicly available and already in a XCSP normalized format. Moreover, we added to the dataset the instances from the MiniZinc suite benchmark. These instances written in FlatZinc [12] were first compiled to XCSP (by using a FlatZinc to XCSP converter provided by the MiniZinc suite) and then normalized following the CSP competition conventions. Unfortunately, since FlatZinc is more expressive than XCSP not all the instances could be successfully converted. The final benchmark was built by considering 7163 CSP instances taken from the Constraint Competition, 2419 CSP instances obtained by the conversion of the MiniZinc instances and then discarding all the instances solved by Mistral during the first 2 seconds computation of the dynamic features. We obtained a dataset containing 4547 instances (3554 from the Constraint Competition and 993 from MiniZinc). For all the instances in the dataset we run all the 22 version of the solvers collecting their results and computation times with a time limit of 1800 seconds (which is the same threshold used in the Constraint Competition). Among the dataset instances, 797 could not be solved by any solver in our portfolio within the time cap. Figure 1a indicates the relative speed of the different solvers by showing, for each solver, the number of instances on which the considered solver is the fastest one. Considering this metric, Mistral is by far the best solver, since it is faster than the others for 1622 instances (36% of the instances of the dataset). In Figure 1b following [17] we show instead the



(a) No. of times a solver is faster (b) Marginal contributions

Fig. 1: Solver statistics

marginal contributions of each solver, that is how many times a solver is able to solve instances that no other solver can solve. Even in this case Mistral is by far the best solver, almost one order of magnitude better than the second one. It is worth noticing that there are also 8 versions of Gecode that do not give a marginal contribution.

4 Methodology

In order to evaluate and compare different portfolio approaches we tested every approach using a 5-repeated 5-fold cross-validation [2]. The dataset was randomly partitioned in 5 disjoint sets called folds. Each of these folds was treated in turn as the test set, considering the union of the 4 remaining folds as training data. In order to avoid a possible overfitting problem (i.e. a portfolio approach that adapts too well on the training data rather than learning and exploiting the generalized pattern) the random generation of the folds was repeated 5 times, thus obtaining 25 sets of instances used to test the portfolio approaches. Every test set was therefore constituted by approximately 909 instances and the portfolio approach for a single fold was built by taking into account (approximately) 3638 training instances. For every instance of every test set we computed the solving strategy proposed by the portfolio approach and we simulated it by using a time cap of 1800 seconds, checking if the solving strategy was able to solve the instance and the time required. To evaluate the performances of the portfolio approach we measured the average solving time (AST) and the percentage of solved instances (PSI) of the portfolio solver, computed on all the 22735 instances of the 25 test sets. In order to present a more realistic scenario, we have considered in the simulation also the time taken to compute the instance features. All the portfolio approaches were tested with portfolios of different sizes. Since we realized that some solvers had a very low marginal contribution we considered portfolios consisting of up to a maximum of 16 solvers. For every size $n = 2, \ldots, 16$ the portfolio composition was computed by using a local search algorithm that maximized the number of instances solved by one of the solvers in

the portfolio. Possible ties were broken by minimizing the average solving time for the instances of the dataset by the solvers in the portfolio.

For the approaches that used off-the-shelf machine learning classification algorithms we used a training set to train a classifier in order to select the best solver among those in the portfolio. For the instances that were not solved by any solver we added a new label *no solver* that could be predicted. For every instance of the test set we simulated the execution of the solver selected by the model. In case the predicted solver was labeled *no solver* or it finished unexpectedly before the time cap the execution of a *backup solver* was simulated for the remaining time. To decide the backup solver, we simulated an election scenario by considering CSPs as voters who have to elect a representative among the 22 candidates solvers. Each CSP could express one or more preferences according to its favorite solver. The election outcomes clearly sustained Mistral as the backup solver since it was the *Condorcet winner*, i.e. the candidate preferred by more voters when compared with every other candidate.

To train the models we used the WEKA tool [5] which implements some of the most well known and widely used classification algorithms. In particular we used a k-nearest neighbors algorithm (IBk), decision trees based algorithms (RandomForest, J48, DecisionStump), bayesian networks (NaiveBayes), rule based algorithms (PART, OneR), support vector machines (SMO), and meta classifiers (AdaBoostM1, LogitBoost). For all the classification algorithms we tried different parameters in order to increase their accuracy. This task was performed following the best practices when they were available or manually trying different parameters starting from the default ones of WEKA. The above approaches based on a ML classification algorithm have been compared against the other approaches described in Section 2.

In order to reproduce the CPHydra approach, we computed the scheduling that it would have produced for every instance of the test set and simulated this schedule. Since this approach does not scale very well w.r.t. the size of the portfolio we were able to simulate this approach only for small portfolios (i.e. containing less than 9 solvers). To compute the PSI and AST we did not take into account the time needed to compute the schedule; therefore the results of CPHydra can be considered only an upper bound of its real performances.

We simulated the SATzilla approach by developing a MATLAB implementation of the cost-sensitive classification model described in [17], with the only exception that ties during solvers comparison are broken by selecting the solver that in general solves the largest number of instances. We employed Mistral as a backup solver in case the solver selected by SATzilla ended prematurely.

To simulate the 3S approach we did not use the original code to compute the static schedule since it is not publicly available. To compute the schedule of solvers we used instead the mixed integer programming solver Gurobi [4] to solve the problem described in [6]. However, in order to reduce the search space, instead of using the column generation method as used by the developers of 3S, we imposed an additional constraint requiring every solver to be run for an integer number of seconds. If the instance was not solved in this time window the solver that solved the majority of the most k-similar instances was used for the remaining time (possible ties were broken by minimizing the average solving time) and, in case of failures, Mistral was used as a backup solver.

Thanks to the code kindly provided by Yuri Malitsky, we were able to adapt ISAC cluster-based techniques to create a solver selector using the "*Pure Solver*

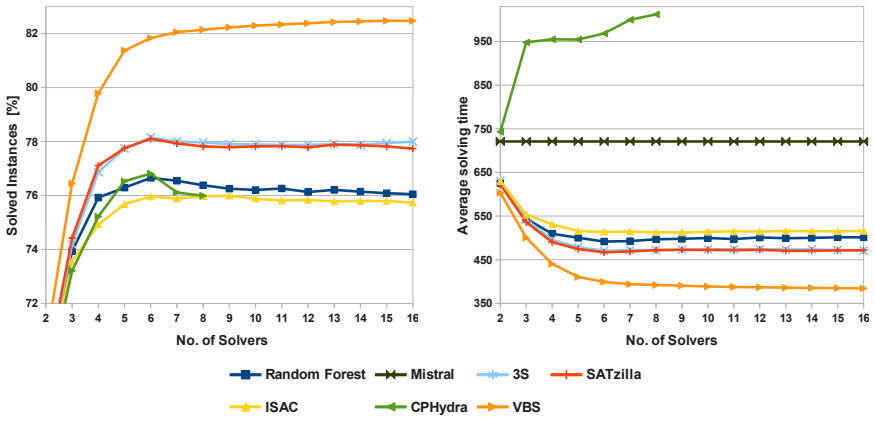


Fig. 2: Performances of portfolio approaches

Portfolio" approach as done for SAT problems in [10]. Also in this case Mistral was used as a backup solver in case of failures of the selected solver. All the code developed to conduct the experiments is available at `http://www.cs.unibo.it/~amadini/cpaior_2013.zip`.

5 Results and Assessments

This section presents the experimental results of our work.

In Fig. 2 for brevity we just show the comparison between the approaches of SATzilla, ISAC, 3S, CPHydra and the best approach using off-the-shelf classifiers which was the one using Random Forest as solver selector (please see [1] for a more extensive comparisons) setting as baselines the performances of Mistral with a time cap of 1800 seconds and of the Virtual Best Solver (VBS), i.e. an oracle that for every instance always chooses the best solver. As already stated, due to the computational cost of computing the schedule of solvers, for CPHydra we report the results obtained using just less than 9 solvers.

It is possible to notice that the best approaches used in SAT, namely 3S and SATzilla, have peak performances. 3S is able to solve usually few more instances than SATzilla (3S have a peak PSI of 78.15% against the 78.1% peak performance of SATzilla) while SATzilla is usually faster (the AST of SATzilla with a portfolio of size 6 was 466.82 seconds against the 470.30 seconds of 3S). Even though conceptually 3S and SATzilla are really different they have surprisingly close performances. This is confirmed also from a statistical point by using the Student's paired t-test with a p-value threshold of 0.05. 3S and SATzilla are instead statistically better than all the other tested approaches for portfolios of size greater than 3 (3S is able to close 26% of the gap of Random Forest w.r.t. the VBS). Moreover, the decay of performances due to the increase of the portfolio size is less pronounced that what usually happens when a classifier is used as a solver selector. As in the classification based approaches, the peak performance was reached with a relatively small portfolio (6 solvers) and the peak

performances of both 3S and SATzilla are statistically significant w.r.t their performances with different portfolios sizes. The performances of ISAC are slightly worse than those of Random Forest: the maximum PSI reached was 75.99% while the Random Forest approach obtained 76.65%.

As far as CPHydra is concerned we saw that it solved the maximum number of instances with a portfolio of size 6 reaching a PSI of 76.81% that was slightly better than the peak performance obtained by Random Forest, even though not in a statistically significant way. After reaching the maximal number of solved instances CPHydra performances are decreasing and in a real scenario they would be rather poor since computing the optimal solvers schedule can consume a lot of time. From Figure 2 it is possible to note that CPHydra differs from other approaches because it is not developed to minimize the average solving time. There is no heuristic to decide which solver needs to be run first in order to minimize the solving time. For this reason, CPHydra is the only approach, among those we have considered, where the PSI and AST values have a positive correlation. Indeed, the Pearson correlation coefficient between PSI and AST values is 0.921, which means that PSI and AST are almost in linear relationship. Conversely for the other best performing approaches the correlation coefficient was always below -0.985 meaning that minimizing the average solving time was like requiring to maximize the number of instances solved and vice versa.

6 Conclusions

In this work we have implemented different portfolio approaches for solving Constraint Satisfaction Problems (CSPs). These approaches have been obtained both by using machine learning techniques and adapting to CSPs other algorithms proposed in the literature, mainly in the SAT solving field. We have evaluated and compared the different approaches by considering a dataset consisting of 4547 instances taken from two different kind of constraint competitions and a selection of 22 versions of different solvers. The portfolio approaches were evaluated on the basis of the number of problems solved and the time taken to solve them. The experimental results show that the approaches that won the last two SAT competitions, namely SATzilla and 3S, are the best ones among those considered in this paper, both for the instances solved and the time needed to solve them. However approaches using off-the-shelf classifiers as solver selector are not that far from the best performances and can potentially be used in scenarios were the time needed to build the model to make the predictions matters. Another interesting empirical fact is that, for all but one the portfolio approaches considered here, there was a strong anti-correlation between the average solving time and the number of solved instances.

We are aware of the fact that our results are not as exhaustive as those existing in the SAT field. However we believe that we made a first step towards a clarification of the importance of the portfolio approaches for solving CSPs. As a future work we plan to extend the number of portfolio approaches by considering also the dynamic schedule approach of 3S [6], the regression based approach of the previous version of SATzilla and other approaches which are not based on feature extraction like [15]. Moreover we are also interested in studying the impact of instance-specific algorithm configuration tools like ISAC or HYDRA [16] in the CSP field by allowing the automatic tuning of search and other solver parameters in order to boost the solver performances.

References

1. R. Amadini, M. Gabbrielli, and J. Mauro. An Empirical Evaluation of Portfolios Approaches for solving CSPs. *ArXiv e-prints*, December 2012. Available at `http://arxiv.org/pdf/1212.0692v1`.
2. Sylvain Arlot and Alain Celisse. A survey of cross-validation procedures for model selection. July 2009.
3. Carla P. Gomes and Bart Selman. Algorithm portfolios. *Artif. Intell.*, 126(1-2):43–62, 2001.
4. Inc. Gurobi Optimization. Gurobi Optimizer Reference Manual, 2012.
5. Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November 2009.
6. Serdar Kadioglu, Yuri Malitsky, Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann. Algorithm Selection and Scheduling. In *CP*, pages 454–469, 2011.
7. Serdar Kadioglu, Yuri Malitsky, Meinolf Sellmann, and Kevin Tierney. ISAC - Instance-Specific Algorithm Configuration. In *ECAI*, pages 751–756, 2010.
8. Zeynep Kiziltan, Luca Mandrioli, Jacopo Mauro, and Barry O'Sullivan. A classification-based approach to managing a solver portfolio for CSPs. In *AICS*, 2011.
9. Kevin Leyton-Brown, Eugene Nudelman, and Yoav Shoham. Learning the Empirical Hardness of Optimization Problems: The Case of Combinatorial Auctions. In *CP*, pages 556–572, 2002.
10. Yuri Malitsky and Meinolf Sellmann. Instance-Specific Algorithm Configuration as a Method for Non-Model-Based Portfolio Generation. In *CPAIOR*, pages 244–259, 2012.
11. Massimo Morara, Jacopo Mauro, and Maurizio Gabbrielli. Solving XCSP problems by using Gecode. In *CILC*, pages 401–405, 2011.
12. Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. Minizinc: Towards a standard cp modelling language. In *CP*, pages 529–543, 2007.
13. Eoin O'Mahony, Emmanuel Hebrard, Alan Holland, Conor Nugent, and Barry O'Sullivan. Using case-based reasoning in an algorithm portfolio for constraint solving. *AICS 08*, 2009.
14. Olivier Roussel and Christophe Lecoutre. XML Representation of Constraint Networks: Format XCSP 2.1. *CoRR*, abs/0902.2362, 2009.
15. Bryan Silverthorn and Risto Miikkulainen. Latent class models for algorithm portfolio methods. In Maria Fox and David Poole, editors, *AAAI*. AAAI Press, 2010.
16. Lin Xu, Holger Hoos, and Kevin Leyton-Brown. Hydra: Automatically Configuring Algorithms for Portfolio-Based Selection. In *AAAI*, 2010.
17. Lin Xu, Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. Evaluating Component Solver Contributions to Portfolio-Based Algorithm Selectors. In *SAT*, pages 228–241, 2012.
18. Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. SATzilla-07: The Design and Analysis of an Algorithm Portfolio for SAT. In *CP*, pages 712–727, 2007.